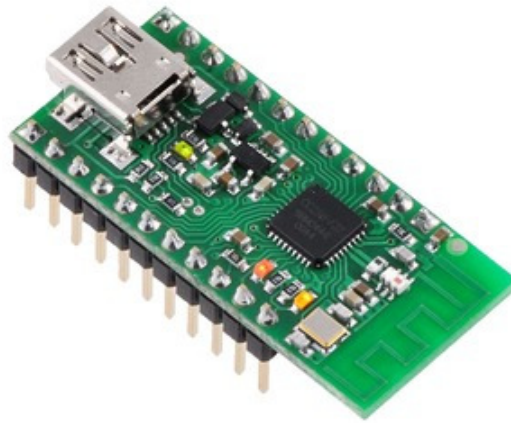


Pololu Wixel User's Guide



1. Overview
 - 1.a. Module Pinout and Components
 - 1.b. Supported Operating Systems
2. Contacting Pololu
3. Getting Started
 - 3.a. Installing Windows Drivers and Software
 - 3.b. Installing Linux Drivers and Software
 - 3.c. Installing Mac OS Drivers and Software
 - 3.d. Loading an Example App
4. Configuring Your Wixels
5. Connecting Your Wixels
 - 5.a. Connecting Power
 - 5.b. Connecting a Microcontroller via TTL Serial
 - 5.c. Connecting Buttons and Starting the Bootloader
6. Using a Virtual COM Port
 - 6.a. Determining the Port Name
 - 6.b. Using a Terminal Program
 - 6.c. Writing PC Software to Use a Serial Port
7. Ensuring a Good Radio Signal
8. Schematic Diagram
9. Wixel Apps
 - 9.a. Example App: Blink LED
 - 9.b. Wireless Serial App
 - 9.c. USB-to-Serial App
 - 9.d. I/O Repeater App
 - 9.e. ShiftBrite App
10. Writing Your Own Wixel App
10. Troubleshooting
 - 10.a. Getting Started in Windows
 - 10.b. Compiling an Example App
 - 10.c. Sharing Your App with the Wixel Community
 - 10.d. USB Configurations Recognized by the Wixel Configuration Software

1. Overview

The Pololu Wixel is a general-purpose programmable module featuring a 2.4 GHz radio and USB. The Wixel is based around the [CC2511F32](#) microcontroller from Texas Instruments, which has an integrated radio transceiver, 32 KB of flash memory, 4 KB of RAM, and a full-speed USB interface. A total of 15 general-purpose I/O lines are available, including 6 analog inputs, and the 0.1" pin spacing makes the Wixel easy to use with [breadboards](#) and perfboards.



Wixel programmable USB wireless module.

We provide free, open-source apps for the Wixel that you can load and configure with its built-in USB bootloader, turning it into whatever you need for your current project. Simply download a different app to reuse the Wixel in your next project.

Our **Wireless Serial app** turns a pair of Wixels into a wireless USB/TTL serial link for communication between two microcontrollers or between a PC and a microcontroller. This can be used, for example, for communication between two robots or to remotely monitor a robot from a computer. Using an RF bit rate of 350 kbps, the serial app is capable of transmitting or receiving up to 10 KB of data per second and can reach a range of approximately 40 feet (under typical conditions indoors). Multiple serial links can be used simultaneously on different channels. Detailed information about the wireless serial app is available in [Section 9.b](#).

Our **USB-to-Serial app** turns a single Wixel in to a USB-to-TTL serial adapter that is capable of baud rates as high as 350,000 bps and supports four serial control signals. This app does not use the radio. Detailed information about this app is available in [Section 9.c](#).

Our **I/O Repeater app** allows you to wirelessly extend the reach of your microcontroller's I/O lines up to 40 feet using two or more Wixels. Detailed information about this app is available in [Section 9.d](#).

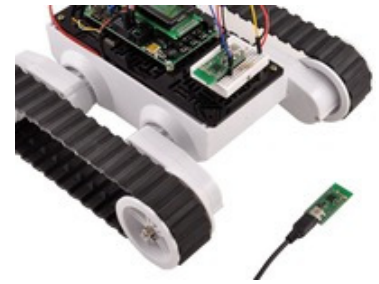
We plan to release additional apps in the future for wireless AVR programming, wireless sensing, and more. You can also write your own apps using the open-source Wixel SDK (see [Section 10](#)) and share them with the community.

Included Hardware

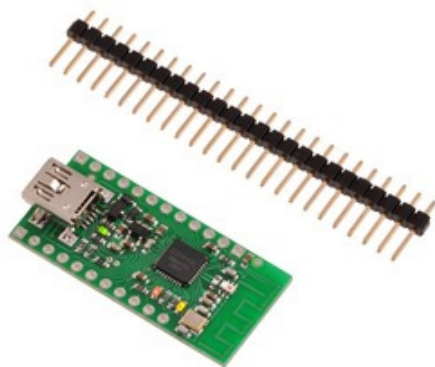
The Wixel is available in two versions:

The [Partial Kit version](#) comes with a 25×1 [straight 0.1" male header strip](#). This version is ideal for compact installations and allows flexibility in choice of connectors.

The [Assembled version](#) comes with its header pins soldered in, so it is ready to be connected to your project with no soldering required.



Wixel programmable USB wireless module enabling wireless communication between a PC and robot.

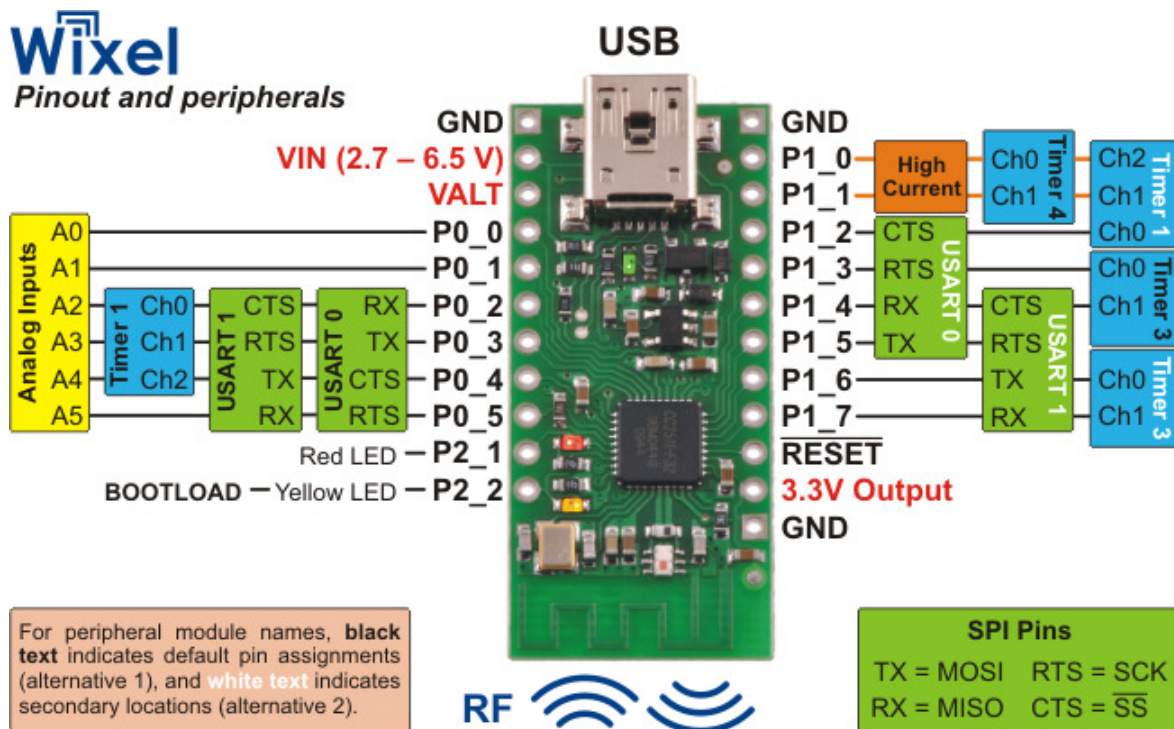


Wixel programmable USB wireless module (without header pins installed).



Wixel programmable USB wireless module (fully assembled).

1.a. Module Pinout and Components



The Wixel can connect to a computer's USB port via a [USB A to mini-B cable](#) (not included). The USB connection is used to configure the Wixel and also to transmit and receive data. The USB connection can also provide power to the Wixel.

On the side of the board opposite the USB connector, the Wixel has a 2.4 GHz PCB trace antenna. This antenna, along with the other RF circuitry, forms a radio that allows the Wixel to send and receive data packets in the 2.4 GHz band. The Wixel is based on the CC2511F32 microcontroller from Texas Instruments, which makes it compatible with the CC2500 transceiver, the CC2510Fx family, and the CC2511Fx family of chips from Texas Instruments. The Wixel's radio is not compatible with Wi-Fi, Zigbee, or Bluetooth. The antenna is a "meandered Inverted F" design that is described in Texas Instrument's application note [AN043](#).

The three **GND** pins are all connected and are at 0 V by definition. When connecting the Wixel to other electronic systems, you should make sure that the Wixel's GND is connected to the other system's GND unless you are doing something very advanced.

The Wixel can be powered from **VIN** pin. Simply connect a 2.7–6.5 V power source between VIN and GND, with the positive terminal going to VIN. It is OK to connect VIN and USB at the same time. See [Section 5.a](#) for more information about powering your Wixels.

The **VALT** pin is connected to three things: the 5V USB bus power from the USB port (through a diode), VIN (through a diode), and to the input of the Wixel's on-board 3.3 V regulator. The connection to 5V is switched off when a power supply is connected to VIN. Most people will not need to use the VALT pin: see [Section 5.a](#) for example uses.

The pin labeled **3V3** on the board (**3.3V Output** in the diagram above) is connected to the output of the Wixel's 3.3V regulator. This power source can be used to power other low-current peripherals in your system. With an input voltage of 5 V (either from USB, VIN, or VALT), this output can provide up to 150 mA of current. At higher input voltages, this output can provide up to 100 mA.

The pin labeled **RST** on the board (**RESET** in the diagram above) is the reset line of the microcontroller. This pin can be driven low to perform a hard reset of the Wixel's microcontroller. This should not be necessary for typical users, but it can be useful while you are developing a Wixel application (see [Section 5.c](#)). This pin is internally pulled high to 3.3 V, so it is okay to leave it unconnected. If you do wire something to this pin, the CC2511F32 datasheet recommends adding an external RC filter with values of 1 nF and 2.7 kΩ close to the pin in order to avoid unintended reset of the microcontroller.

The Wixel has 15 free I/O lines whose behavior depends on the application that is loaded onto the Wixel. Specifically, these are all of the pins on Port 0 (P0_0 through P0_5), all of the pins on Port 1 (P1_0 through P1_7), and P2_1. The **P2_1** pin is tied to the red LED but the other 14 free I/O lines are only connected to the microcontroller. The **P2_2** line is also accessible, but it is tied to the yellow LED and is used to get the Wixel in to bootloader mode (see [Section 5.c](#)).

The amount of current that can be supplied by the CC2511F32's I/O pins is not well-documented by the manufacturer. According to [this forum post by a TI Employee](#), regular I/O pins are designed to be able to source 4 mA while **P1_0** and **P1_1** are designed for 20 mA.



Caution: The Wixel's I/O lines are **not** 5V tolerant. You must use level-shifters, diodes, or voltage dividers to connect the Wixel to outputs from 5V systems.

The CC2511F32 has several peripherals that are available to be used in Wixel applications:

- 2 USARTs which can perform asynchronous serial or SPI communication
- 3 timers that are capable of PWM output as shown above, plus 1 more internal timer
- 6 analog input-capable pins, connected to a 7–12 bit ADC

Different Wixel applications may use different sets of these peripherals. Consult the application documentation for details on the behavior of the I/O lines.

The pinout and peripheral diagram at the top of this section is also available as a [printable pdf \(145k pdf\)](#).

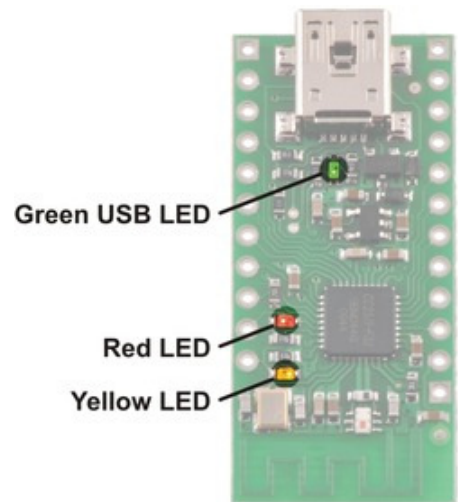
The Wixel has three indicator LEDs:

Green USB LED

The green LED is powered from USB, so it can only be turned on when USB cable is connected and supplying power to the Wixel.

While the Wixel is in bootloader mode (i.e. the app is stopped), this LED is used to indicate the USB status of the device. When the Wixel USB Bootloader connects to USB, the green LED starts blinking slowly. The blinking continues until the bootloader receives a particular message from the computer indicating that the Wixel USB Bootloader drivers are installed correctly (see [Section 3.a](#) for driver installation instructions). After the bootloader gets this message, the green LED will do a double-blinking pattern. The green LED also turns off during USB Suspend Mode, which happens when the computer goes to sleep or shuts down the USB port for any other reason.

While the Wixel is running its app, the behavior of the LED depends on the app. The standard apps provided by Pololu all behave as follows: When the app connects to USB, the green LED starts blinking slowly. The blinking continues until the app receives a particular message from the computer indicating that the app's drivers are installed correctly. After the app gets this message, the green LED turns solidly on. The green LED also turns off during USB Suspend Mode, which happens when the computer goes to sleep or shuts down the USB port for any other reason.



Wixel indicator LEDs.

Red LED

While the Wixel is in bootloader mode (i.e. the app is stopped), this LED indicates whether there is an application on the Wixel. If there is no application, the red LED will be on. Otherwise, it will be off. By default, the Wixel does not have an application on it, so this LED will be on the first time you power your Wixel.

The **P2_1** pin is connected to the red LED, so this line will go high when the red LED is on and otherwise be pulled low.

While the Wixel is running its app, the behavior of this LED depends on the app. See the documentation of your particular app for more details.

Yellow LED

While the Wixel is in bootloader mode (i.e. the app is stopped), this LED turns solidly on and flickers whenever the bootloader receives a command from Wixel software on the computer. The Wixel Configuration Utility queries the state of the bootloader once per second, so if the Wixel Configuration Utility is open then the LED will flicker once per second. While the Wixel is being programmed, the yellow LED will constantly flicker.

The **P2_2** pin is connected to the yellow LED, so this line will go high when the yellow LED is on and otherwise be pulled low.

While the Wixel is running its app, the behavior of this LED depends on the app. See the documentation of your particular app for more details.

1.b. Supported Operating Systems

The **Wixel USB drivers and configuration software** currently work under Windows 7, Windows Vista, and Microsoft Windows XP (SP 3). We plan to support Linux and Mac OS in the future.

Any **Wixel app** that implements a single USB virtual COM port will work in Mac OS or Linux with no special driver installation required, so you could configure your Wixel using Windows but then use it on a Mac or Linux computer.

2. Contacting Pololu

We would be delighted to hear from you about any of your projects and about your experience with the Wixel.

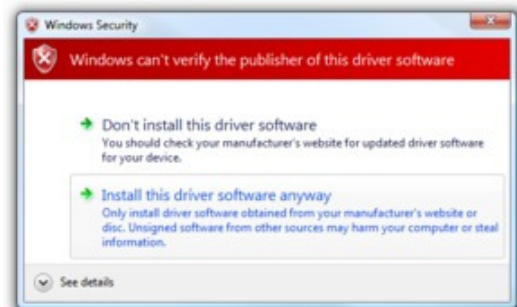
You can [contact us](#) directly or post on our [forum](#). Tell us what we did well, what we could improve, what you would like to see in the future, or anything else you would like to say!

3. Getting Started

3.a. Installing Windows Drivers and Software

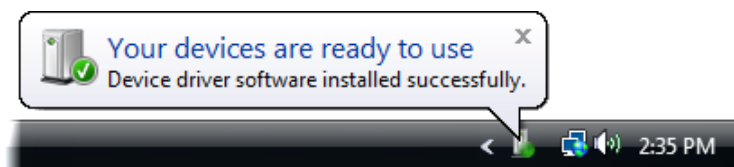
Before you connect a Wixel to a computer running Microsoft Windows, you should install the drivers:

1. Download the [Wixel Windows Drivers and Software](#) (12007k zip)
2. Open the ZIP archive and run *setup.exe*. The installer will guide you through the steps required to install the Wixel Configuration Utility, the Wixel command-line utility (WixelCmd), and the Wixel drivers on your computer. If the installer fails when run directly from the ZIP file, extract the contents of the ZIP file to a temporary directory on your computer, right click *setup.exe*, and select “Run as Administrator”.
3. During the installation, Windows will warn you that the drivers have not been tested by Microsoft and recommend that you stop the installation. Click “Continue Anyway” (Windows XP) or “Install this driver software anyway” (Windows 7 and Vista).



4. After the installation is finished, your start menu should have a shortcut to the *Wixel Configuration Utility* (in the *Pololu* folder). This is a Windows application that allows you to load apps onto your Wixel. There will also be a command-line utility called *WixelCmd* which you can run at a Command Prompt.

Windows 7 and Windows Vista users: Your computer should now automatically install the necessary drivers when you connect a Wixel. No further action from you is required.

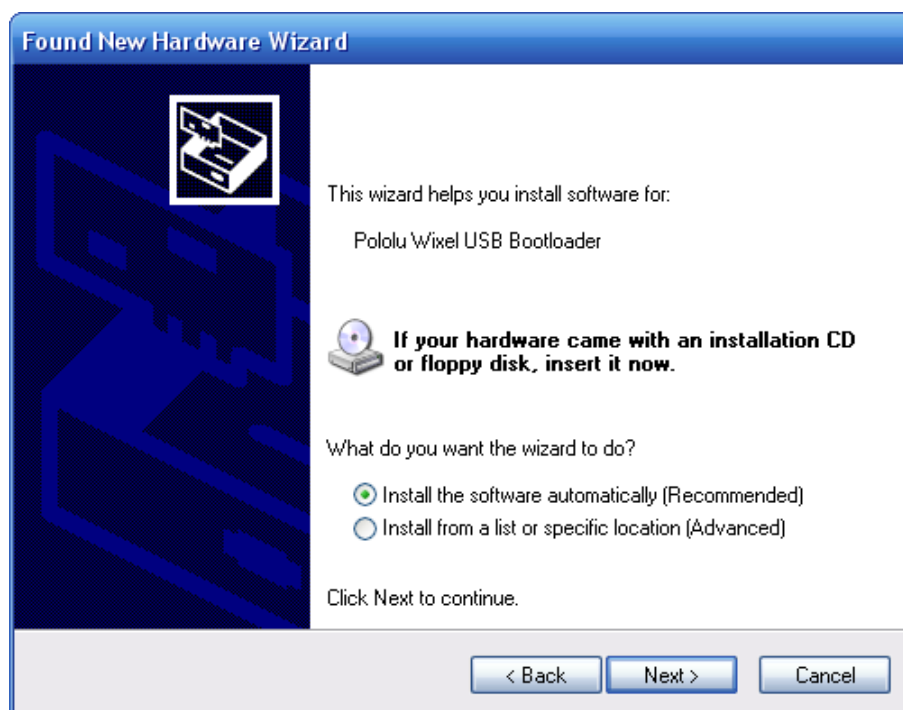


Windows XP users: Follow steps 5–9 for each new Wixel you connect to your computer. You will also have to follow these steps again the first time you run an actual Wixel app on the Wixel.

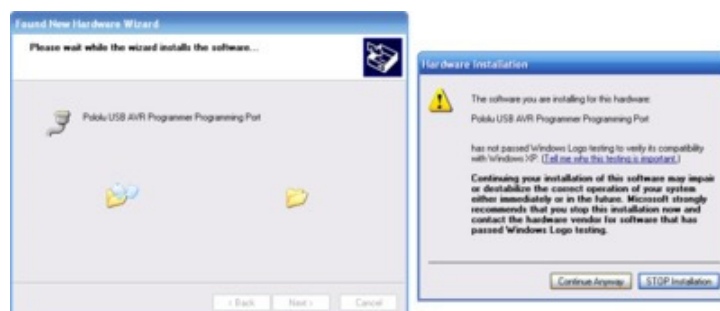
5. Connect the device to your computer's USB port.
6. When the “Found New Hardware Wizard” is displayed, select “No, not this time” and click “Next”.



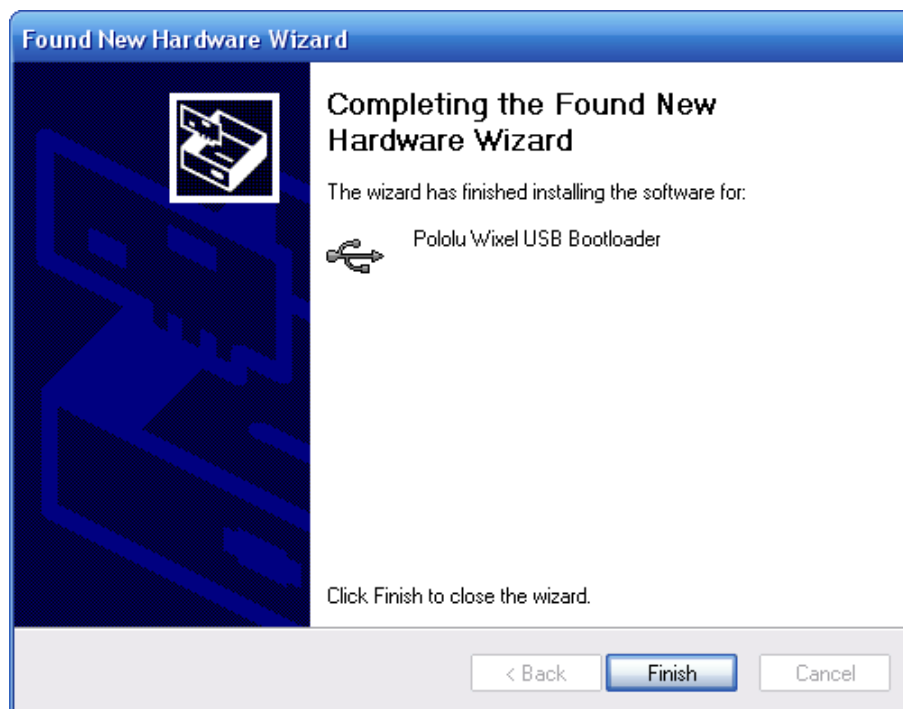
7. On the second screen of the “Found New Hardware Wizard”, select “Install the software automatically” and click “Next”.



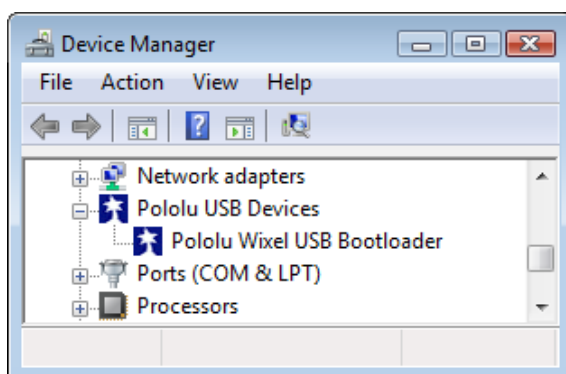
8. Windows XP will warn you again that the driver has not been tested by Microsoft and recommend that you stop the installation. Click “Continue Anyway”.



9. When you have finished the “Found New Hardware Wizard”, click “Finish”.



After installing the drivers, if you go to your computer's Device Manager and expand the "Pololu USB Devices" list, you should see an entry for the Pololu Wixel USB Bootloader.

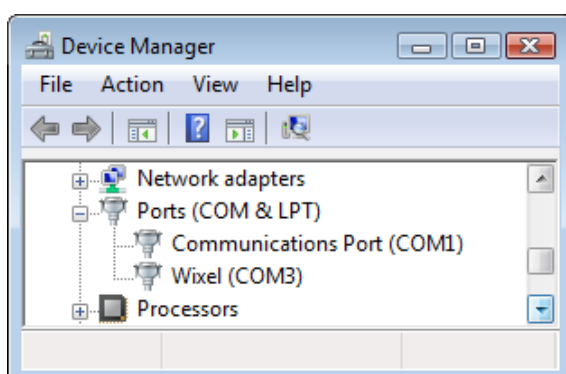


Windows Vista or Windows 7 Device Manager showing a Wixel in bootloader mode.

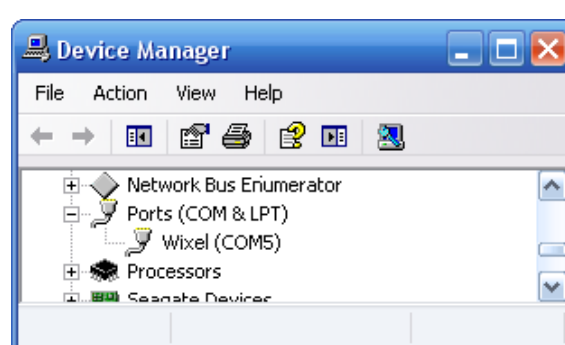


Windows XP Device Manager showing a Wixel in bootloader mode.

If you see the "Pololu Wixel USB Bootloader" entry in your device manager, it means that your Wixel is in bootloader mode. Your Wixel should go in to bootloader mode when you first plug it in to USB, because there is no app on the Wixel by default. Once you have loaded an app onto the Wixel using the Wixel Configuration Utility, and the app is running, then you will **not** see the Pololu Wixel USB Bootloader entry in your Device Manager. The entry you see in the Device Manager will depend on the application that is loaded on the Wixel. Some apps might not enable the USB interface, in which case you will see no entry for the Wixel in the Device Manager. However, typical Wixel Apps will appear in your Device Manager as a single Virtual COM port (with product ID 0x2200) in the "Ports (COM & LPT)" list as shown below:



Windows Vista or Windows 7 Device Manager showing a Wixel that is running an app with a virtual COM port.



Windows XP Device Manager showing a Wixel that is running an app with a virtual COM port.

In parentheses, you will see the name of the port (for example, COM5 or COM6). Some software will not allow connection to higher COM port numbers. If you need to change the COM port number assigned to a Wixel, you can do so using the Device Manager. Bring up the properties dialog for the COM port and click the “Advanced...” button in the “Port Settings” tab. From this dialog you can change the COM port assigned to your device. Windows will remember which COM port was assigned to which Wixel using the built-in serial number of the Wixel; a given Wixel will always get assigned to the same COM port regardless of which USB port it is plugged in to.

3.b. Installing Linux Drivers and Software

The Wixel software currently does not work on Linux, but certain Wixel apps should. See [Section 1.b](#) for details.

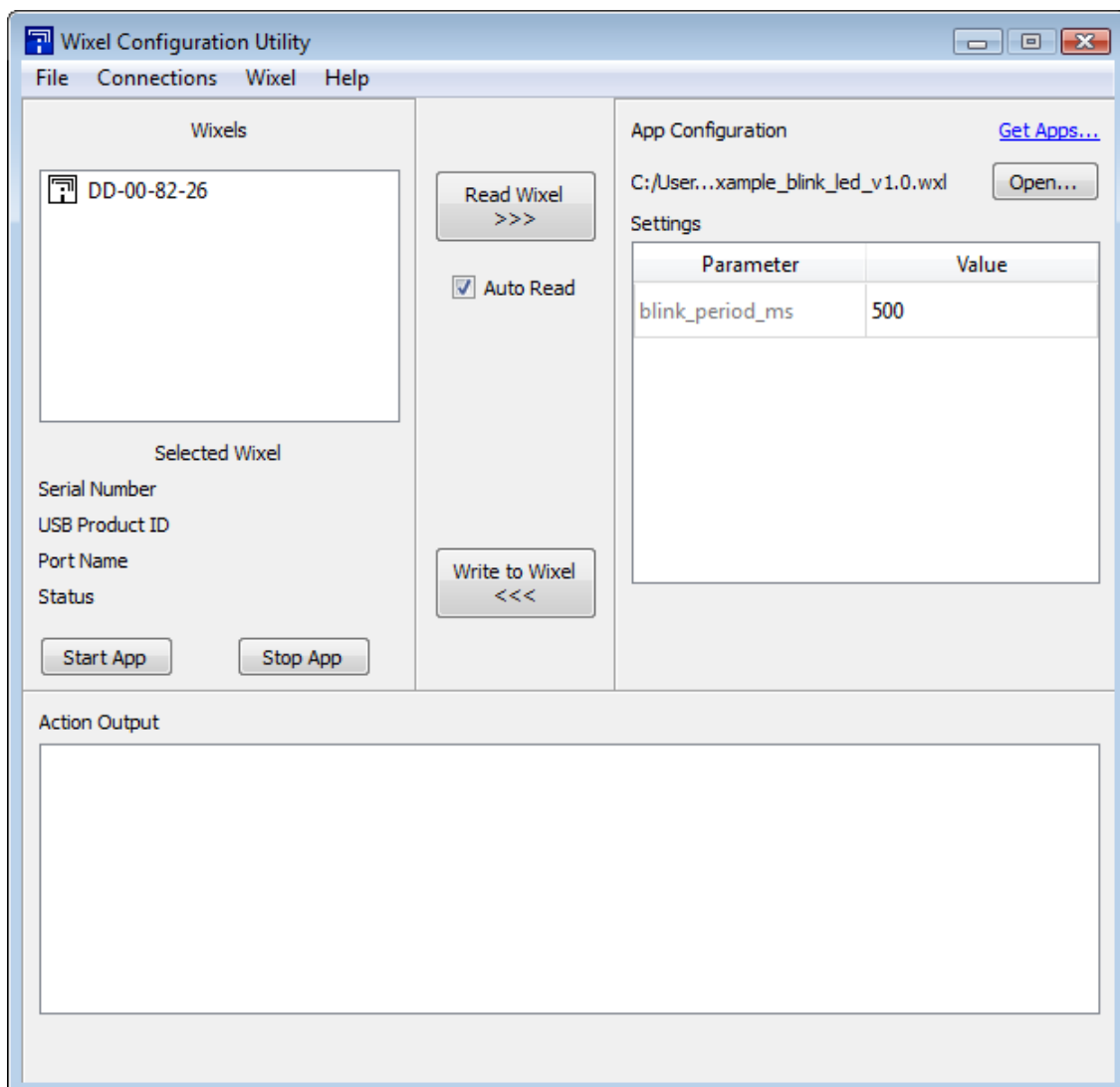
3.c. Installing Mac OS Drivers and Software

The Wixel software currently does not work on Mac OS, but certain Wixel apps should. See [Section 1.b](#) for details.

3.d. Loading an Example App

When you first get your Wixel it will have no application loaded. To make your Wixel do something useful, you must load an app onto it. This section guides you through the steps needed to load an example application onto the Wixel using the Wixel Configuration Utility.

1. Install the Wixel drivers and software by following the instructions in the preceding sections.
2. Download the example application here: [Example Blink LED App v1.0](#) (11k wxl). If you want to see the source code, it is in the Wixel SDK under apps/example_blink_led. (See [Section 10.a](#).)
3. Open the app in the Wixel Configuration Utility. To do this in Windows, you can simply double-click on the Wixel App (WXL) file. Alternatively, you can open the Wixel Configuration Utility, click the “Open...” button, and select the Wixel App file. In Windows, you can find the Wixel Configuration Utility in the Pololu folder in your Start Menu.
4. Connect a Wixel to your computer via USB. You should see it appear in the “Wixels” list. If it does not appear, you might need to use a button or wire to get your Wixel into bootloader mode (see [Section 5.c](#)). At this point, your screen should look something like this:



The Wixel Configuration Utility with the Example Blink LED App open.

5. Note that in the Wixels box, there is a list of all the Wixels connected to USB that the Wixel Configuration Utility can recognize. There is one Wixel

connected, and its 32-bit serial number is displayed in the list. Also note that in the App Configuration box, we have opened the example_blink_led_v1.0.wxl app. This app has one parameter, blink_period_ms, and it is currently set to 500 (the default).

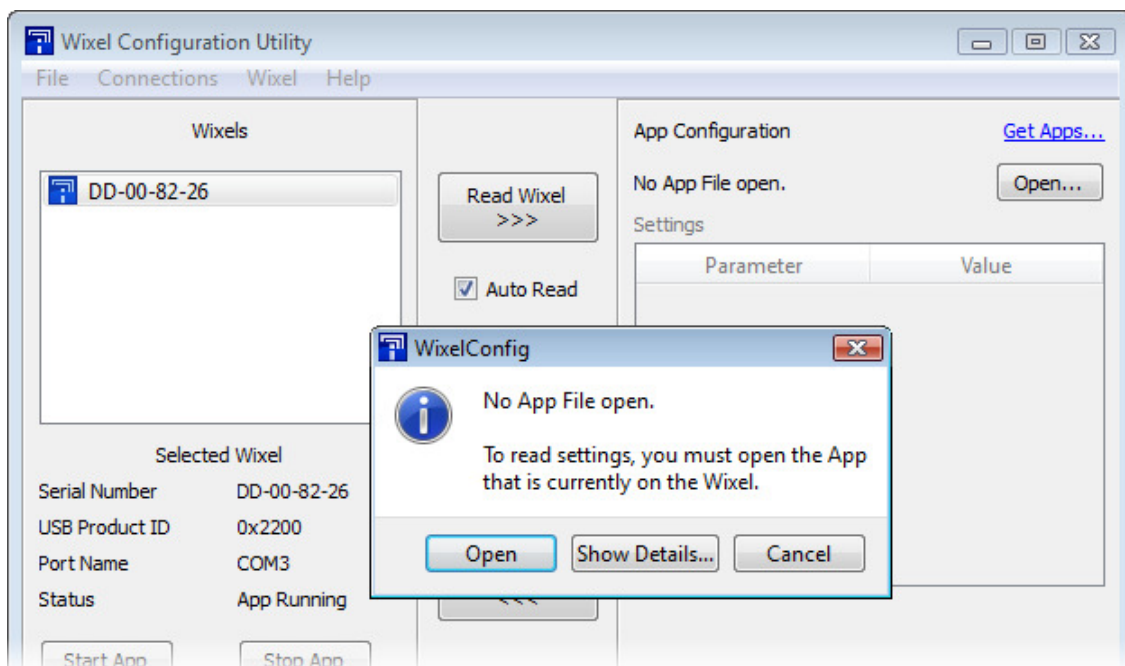
6. Select the Wixel by left-clicking on its entry in the Wixels list. If you see a dialog box pop up, this is because the Wixel already has an application on it and the Auto Read checkbox is checked. Click the “Cancel” button in that dialog because we are not interested in reading the contents of the Wixel yet.

7. Click the **Write to Wixel** button. This writes the currently open app and the settings to the selected Wixel, and then starts running the application.

8. The example blink LED application should be running now. The Wixel’s yellow LED should be off, and the red LED should be blinking. If you are in Windows XP and this is the first time you have run an application on this Wixel, the Found New Hardware Wizard will pop up and you will have to follow steps 5–9 from [Section 3.a](#) to install the drivers properly. After the USB drivers are installed properly, the green LED should be on solid. Congratulations, you have successfully configured your Wixel!

9. The speed of the blinking is determined by the blink_period_ms parameter. The units of this parameter are milliseconds (ms). Try changing blink_period_ms to 100 by double-clicking on the number and typing “100”. You can now write the new configuration to the Wixel by clicking the “Write to Wixel” button. After the writing is done and the app is running, you should see the red LED blinking 5 times faster than it was before.

10. The Wixel Configuration Utility can also read the settings from the Wixel. To demonstrate this, close the Wixel Configuration Utility, reopen it, and select your Wixel. Since the Auto Read checkbox is checked by default, and there is an application on the Wixel, the Wixel Configuration Utility will attempt to read the Wixel’s settings. If you have not yet opened the correct app, a dialog box like the one below will pop up:



If Auto Read is enabled when you select a Wixel with an app on it, then you might be prompted you to open the App File.

11. To read the settings from a Wixel, you will need to open the app that is currently on the Wixel. Click the “Open” button in the dialog, and select example_blink_led_v1.0.wxl.

It is necessary to open the app file that you originally loaded onto the Wixel because the file contains metadata required to interpret the settings contained in the Wixel’s flash memory. If you open a different app file, even a different version of the same app, your settings will likely be corrupted. In this case, a warning dialog box will pop up to warn you and give you some options. Since you still have the correct app file, you should not see that dialog now.

12. After you open the Wixel App file, the Wixel Configuration Utility will read the Wixel and compare its contents to what is in the app. You will then see the Wixel’s settings displayed on the right: blink_period_ms should be 100. Note that the number 100 is displayed in bold. This is because it differs from the default setting, which is 500. You can reset it to the default at any time by right-clicking on the number and selecting “Reset to Default Value”.

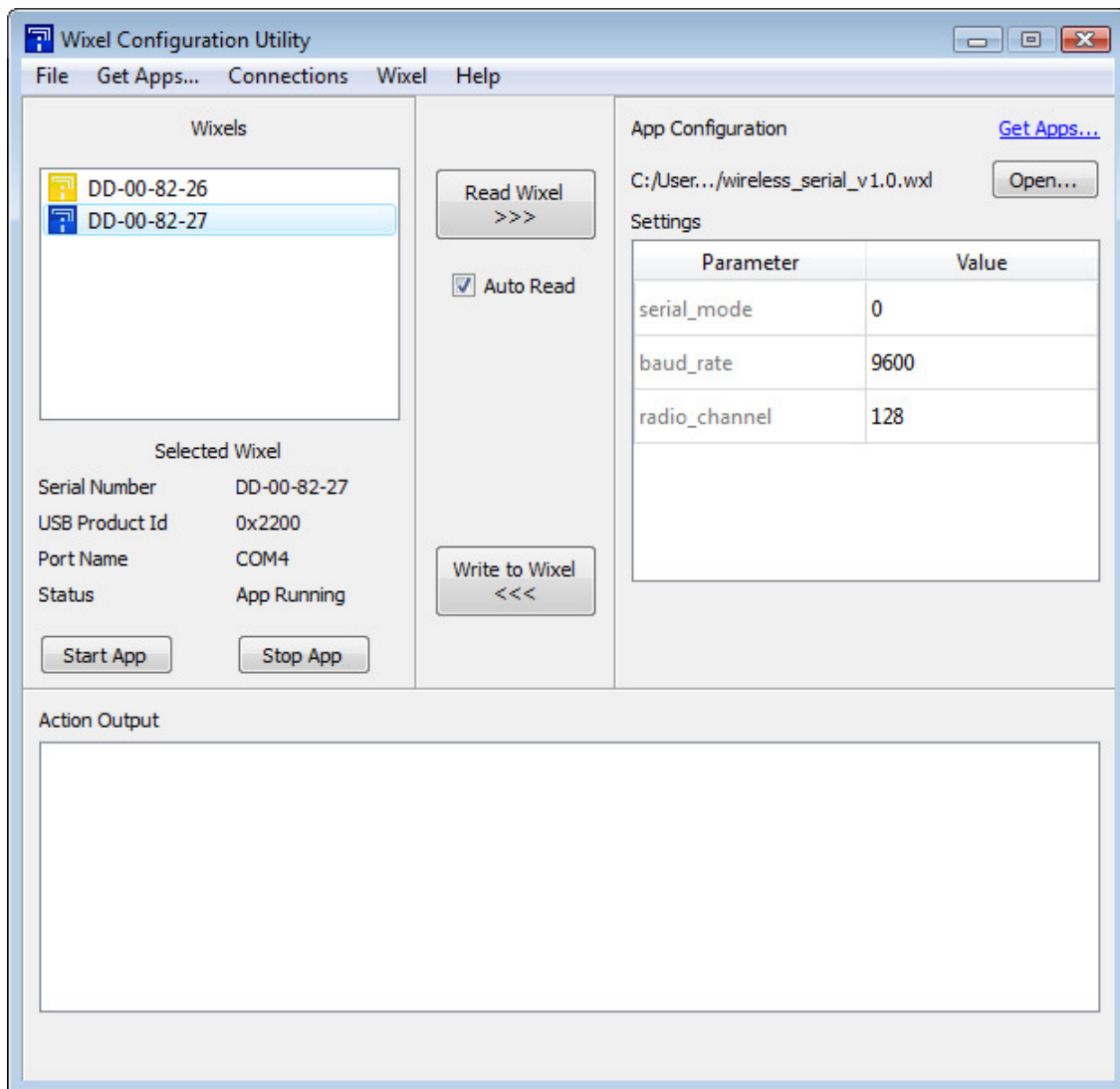
After completing this tutorial you should be comfortable with writing apps to the Wixel and reading back the settings. This is all you need to know in order to configure your Wixels. When you load a real application, such as the [Wireless Serial App](#), the only thing that will be different are the names and meanings of the parameters. To understand what the different parameters mean, refer to the documentation for your specific app.

Some apps might implement a non-standard USB interface (or no USB interface at all). In that case, they will not be recognized by the Wixel Configuration Utility while the app is running, so you will need to get them in to bootloader mode some other way (see the app’s documentation and also [Section 5.c](#)).

4. Configuring Your Wixels

The Wixel Configuration Utility allows you to write and read settings from the Wixel. This section explains all of the features of the Wixel Configuration

Utility in detail.



The Wixel Configuration Utility with 2 Wixels connected and an App file open.

Wixels

On the left side of the window, in the Wixels box, you can see a list of all the Wixels connected to the computer that are recognized by the Wixel Configuration Utility. The Wixel Configuration Utility should recognize any Wixel that is in bootloader mode (i.e. the app is stopped) or is running an app that implements a single USB Virtual COM port with a USB Vendor ID of 0x1FFB (for Pololu) and a Product ID of 0x2200. If your Wixel is connected to your computer, but does not show up in the Wixel Configuration Utility, then your drivers might not be installed properly, or the Wixel might be running an application that uses a different type of USB interface or doesn't use USB at all. If you have trouble getting the Wixel Configuration Utility to recognize your Wixel, then see [Section 10](#) for help.

The text displayed in the Wixel list (e.g. "07-C2-C8-3A") is the serial number of your Wixel. Each Wixel has a unique 32-bit serial number which was randomly generated and assigned to it when the Wixel was manufactured.

The icon displayed in the Wixel list represents the current state of the Wixel. Each Wixel will be in one of these states:

Wixel Status	Icon	Description
App Running		The app you loaded on the Wixel is now running.
App Stopped		The app you loaded on the Wixel is currently stopped; the Wixel is in bootloader mode.
No App		There is no app on the Wixel; the Wixel is in bootloader mode.
Reconnecting		The Wixel is reconnecting, disconnecting, or in a transitional state.

If you select a Wixel, you can see more information about it in the area below the list. The **USB Product ID** is the current product ID presented by the Wixel on its USB interface, as defined in the USB Specification. The **Port Name** is the name of the virtual COM port that has been assigned to the Wixel. In Windows, the Port Name is also available in the Device Manager.

The **Stop App** button stops the application that is running on the currently-selected Wixel, putting that Wixel in to bootloader mode. The **Start App** button takes the Wixel out of bootloader mode to run the application that is currently on it.

App Configuration

On the right side of the window, in the App Configuration box, you can see the name of the currently-open app and the current settings.

You can open a different app by clicking the “Open...” button. In Windows, you can also open an app simply by double-clicking on it. The Wixel Configuration Utility can open app files in either the WXL format (documented in 0J603) or the standard [Intel HEX](#) format.

You can change the current settings by double-clicking on a value and typing a new value in. The parameters that are available depend on the app that is open; different apps have different parameters available.

Please see the documentation for your specific application for an explanation of what the parameters mean, and what the valid values are. The Wixel Configuration Utility will **not** prevent you from entering invalid or inconsistent values.

Writing to a Wixel

After you have chosen the app and settings you want to use, and selected a Wixel, you can write the app and settings to the Wixel by clicking the **Write to Wixel** button. This will erase whatever application was previously on the Wixel and write the new application to the Wixel. When the write operation is done, the Wixel will be restarted and the application should start running.

Reading from a Wixel

To read the settings from a Wixel that has been programmed, select the Wixel. If the **Auto Read** checkbox is checked, then the Wixel will automatically be read. Uncheck this box if you want to retain current settings when changing Wixels (for example, when you want to write the same app and settings to multiple Wixels). If the box is unchecked, you can click the **Read Wixel** button at any time to read settings from the selected Wixel.

To read the settings from a Wixel, you will need to open the app that is currently on the Wixel. This is necessary because the app file contains metadata which is needed in order to correctly interpret the settings contained in the Wixel’s flash memory.

If you have lost the app file and want to read the contents of your Wixel, select **Read Flash and Export to HEX File...** from the Wixel menu. You can then open the exported HEX File as an App and use it to program other Wixels. The settings from the Wixel will be contained in the exported HEX file but you will not be able to read these settings in the Wixel Configuration Utility.

If the Wixel’s application is running when the read operation starts, the Wixel Configuration Utility will temporarily stop the application and put the Wixel in to bootloader mode in order to read its contents. When the read operation is completed, the Wixel Configuration Utility will restart the app.

Other Commands

The **Erase Wixel** command (found in the Wixel menu) erases the app from the currently selected Wixel (every bit in the application flash section becomes a 1).

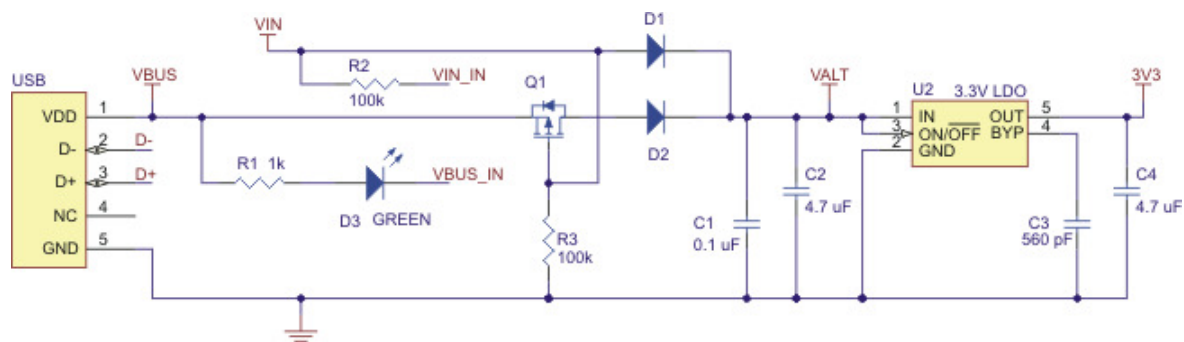
The **Verify Wixel** command reads the currently selected Wixel and tells you whether its contents are identical to the app and settings displayed on the right in the App Configuration box.

5. Connecting Your Wixels

This chapter explains some of the electrical connections you might need to make to get your Wixel working the way you want it to.

5.a. Connecting Power

The two main ways of powering the Wixel are the USB port and the VIN pin. The schematic of the Wixel’s power system is shown below:



Wixel power system schematic diagram.

VIN Power Input

The Wixel can be powered from VIN if you connect a 2.7–6.5 V power supply (e.g. battery or regulator) to the GND and VIN pins. The negative terminal should be connected to GND. The positive terminal should be connected to VIN. It is okay to have both USB and VIN connected at the same time.

USB Power Input

The Wixel can be powered from USB if you connect a USB cable and leave VIN disconnected. The Wixel will draw its power from USB if VIN is disconnected or it is below about 4 V.

3V3 Power Output

The Wixel's 3V3 pin gives access to the output of the Wixel's 3.3 V regulator. If the Wixel's power supply drops below approximately 3.5 V, the 3V3 output will be less than 3.3 V. Normally this output can provide up to 150 mA, but if the Wixel's power supply is above 5 V then it is limited to 100 mA. You can use 3V3 to power your own 3.3 V devices.

VALT Power Output

VALT provides access to the input pin of the Wixel's 3.3 V regulator, which is connected through a diode to VIN or to the USB bus voltage, depending on which power source is connected. You can use VALT to power your own circuits as long as you do not draw more than about 500 mA.

For example, if you leave VIN disconnected, VALT can power 5V devices.

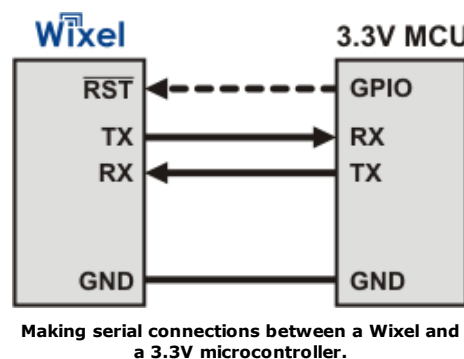
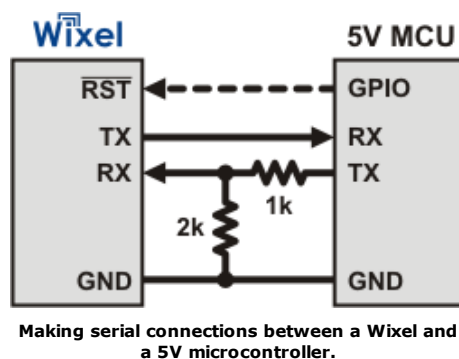
Low Power Considerations

The CC2511F32 is capable of a sleep mode (PM2) where the chip draws less than 1 μ A and is still capable of waking itself up. Without modifying the Wixel's hardware, it is possible to power it from VIN and get the current consumption down to around 100 μ A. Most of that current is consumed by the Wixel's 3.3 V regulator. To get rid of it, you will need to sever the output pin of the regulator and power the Wixel directly from the 3V3 pin with a 2.0–3.6 V power supply. For details on how to do this, please contact Pololu.

Please note that currently none of the Wixel apps support low power modes, so the Wixel will draw approximately 30 mA at all times. To make your Wixel operate with low power you would have to write your own app or modify one of the existing apps. You will need to make sure that all of the I/O lines are either outputs or get pulled high or low: an input line at an intermediate voltage can consume several microamps of extra current.

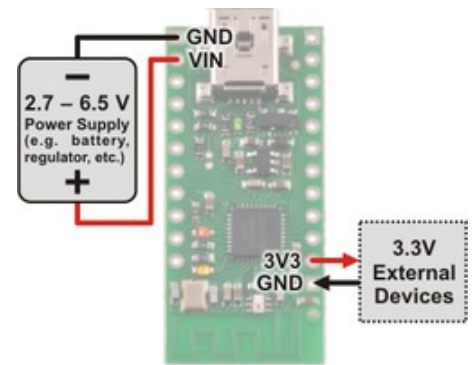
5.b. Connecting a Microcontroller via TTL Serial

If you have loaded a Wixel app that employs one of the Wixel's two UARTs (such as the [Wireless Serial App](#)), then the Wixel can send and/or receive asynchronous TTL serial bytes from a microcontroller.

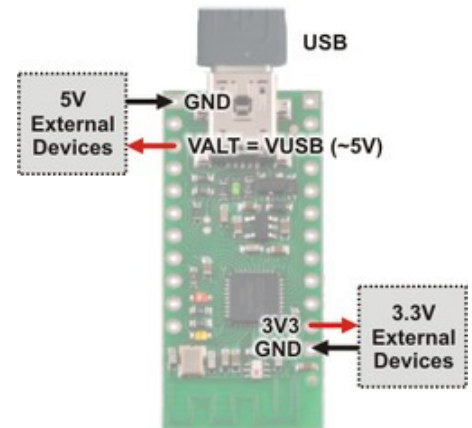


To connect your microcontroller to a Wixel for serial communication, make these connections:

- **GND:** Connect the ground (also known as GND or VSS) of your microcontroller to one of the GND pins on the Wixel. This connection is required.
- **TX:** If you want the microcontroller to be able to receive serial bytes from the Wixel, connect the Wixel's TX line to the microcontroller's RX line.
- **RX:** If you want your microcontroller to be able to send serial bytes to the Wixel, connect the microcontroller's TX line to the Wixel's RX line. The Wixel's RX line is **not** 5 V tolerant. If your microcontroller is running at 5 V (or any voltage significantly above 3.3 V) then you will need to add extra components to ensure that your microcontroller never drives the Wixel's RX line higher than 3.3 V. A simple voltage divider consisting of 2 resistors as shown in the diagram above will suffice. The Wixel's RX line has an internal 20 k Ω pull-up resistor.
- **RST:** If you want the microcontroller to be able to reset the Wixel, then connect the Wixel's $\overline{\text{RST}}$ line to any free general-purpose I/O (GPIO) line on the microcontroller. The microcontroller can drive this line low to reset the Wixel and then stop driving the line to release the Wixel from reset. The Wixel's RST line is **not** 5 V tolerant. If your microcontroller is running at 5 V (or any voltage significantly above 3.3 V) then you must avoid driving the



The Wixel can be powered from an external power source connected to VIN.



The Wixel can be powered from USB.

Wixel's RST line high. If that is not possible, then you could put a diode between the Wixel's $\overline{\text{RST}}$ line and the microcontroller's GPIO to prevent current from flowing in the wrong direction (from the GPIO). The connection to $\overline{\text{RST}}$ is optional and not required for sending or receiving data.

Please refer to the documentation of your specific Wixel app to determine the location of the TX pin(s) and RX pin(s).

5.c. Connecting Buttons and Starting the Bootloader

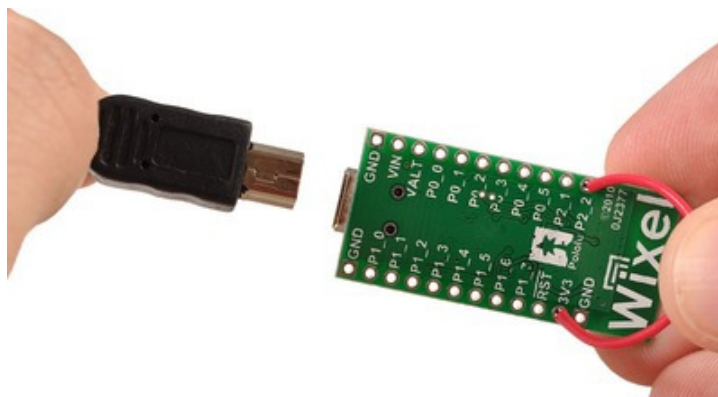
In order to load a new app or new settings onto your Wixel (or read the Wixel's flash memory) you will need to get it in to bootloader mode. Most Wixel apps support a special USB command for putting the Wixel in to bootloader mode and the Wixel Configuration Utility can send that command automatically when you try to access the Wixel's flash. However you may find yourself in a situation where that method will not work. This can happen for two reasons:

- You accidentally loaded a malfunctioning program onto the Wixel that is incapable of responding to the special USB command.
- You loaded a program which uses a different type of USB interface or no USB interface. In this case, check the documentation of the the app to see if there is a convenient way for getting the Wixel in to bootloader mode.

No matter what state the Wixel is in, you can force it to go in to bootloader mode by having USB connected and P2_2 connected to 3V3 at the moment when the microcontroller starts running.

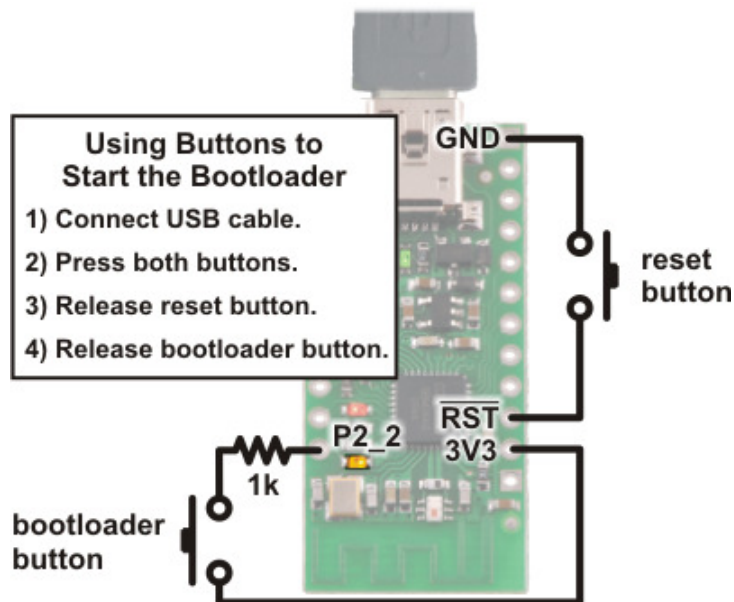
There are two main ways to accomplish this.

One way is to disconnect the Wixel from any possible power sources, connect P2_2 to 3V3 using a wire, and then plug it in to USB.

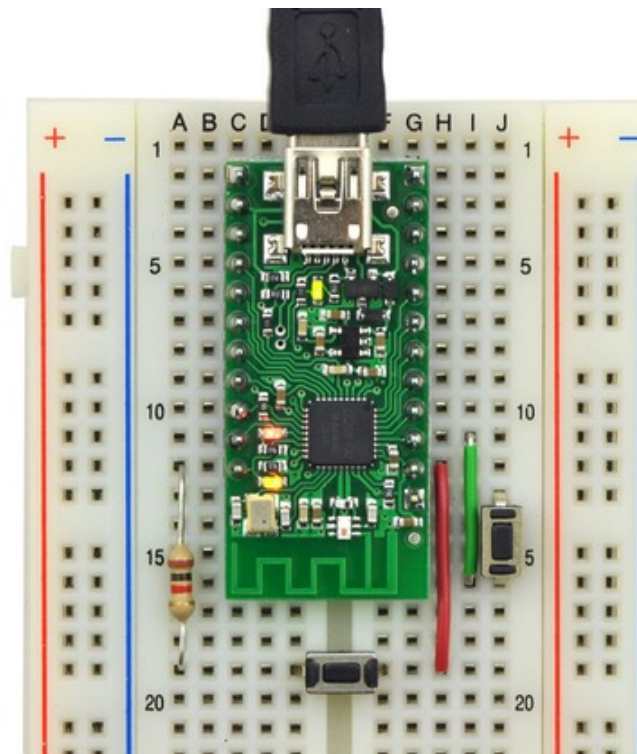


Using a wire to put the Wixel into bootloader mode.

Another way to is to wire a bootloader button and a reset button to the Wixel and follow the procedure shown in the picture below:



Using pushbuttons to put the Wixel into bootloader mode.



Wixel on breadboard with a bootloader button and reset button connected.

6. Using a Virtual COM Port

Most of the available Wixel apps implement a USB interface that consists of a single virtual COM (serial) port. This interface allows you to send and receive bytes from the Wixel in the same way you would send and receive bytes from any other serial port on your computer.

6.a. Determining the Port Name

To connect to a COM port, you usually have to know the name of the port.

In Windows, the port name will be something like “COM4” and you can determine the port name by selecting the Wixel in the Wixel Configuration Utility and looking at the “Port Name” property displayed below. You can also find out the port name by looking the “Ports (COM & LPT)” list in your Device Manager.

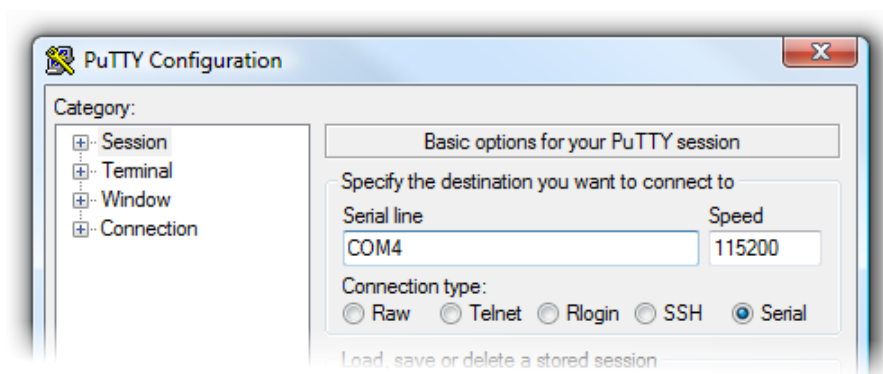
Windows Tip: Besides having names like “COM5” and “COM6”, the virtual COM ports provided by the Wixel also have names like “\\.\USBSER000” and “\\.\USBSER001”. These names are assigned sequentially whenever a device with a virtual COM port is plugged in. If you only have one device with a virtual COM port plugged in to your computer, the name “\\.\USBSER000” will usually be assigned to it. These names will work with most programs that allow you to specify arbitrary port names.

6.b. Using a Terminal Program

There are many free terminal programs available which are capable of sending and receiving bytes on a virtual COM port. These programs include [PuTTY](#) (Windows or Linux), [Tera Term](#) (Windows), and [Br@y Terminal](#) (Windows). Advanced users developing scripted applications may prefer the free terminal program [kermit](#). To use any of these terminal programs with the Wixel, you must specify the port name (see [Section 6.a](#)) and the baud rate. The baud rate may or may not affect anything; see your application’s documentation. The characters you type will be transmitted on the programmer’s **TX** line. Bytes received by the programmer on the **RX** line will be displayed on the screen by the terminal program.

Typical terminal programs will allow you to choose several other settings besides the baud rate. If you are not sure what settings to use, then you should pick 8 data bits, 1 stop bit, no parity, and no flow control.

Typical terminal programs will not allow you to use the serial control signals, but Br@y terminal does. You can click the “DTR” and “RTS” buttons to change the state of the DTR and RTS signals. The state of the CTS, CD, DSR, RI, DTR, and RTS signals are indicated by the colors of the corresponding buttons.



PuTTY is a free Windows terminal program that can send and receive bytes on a serial port.

If you need to send and receive non-ASCII bytes, you can use the [Pololu Serial Transmitter Utility for Windows](#) or Br@y Terminal.

6.c. Writing PC Software to Use a Serial Port

You can write your own computer program that communicates with a serial port. The freely available Microsoft .NET framework contains a [SerialPort](#) class that makes it easy to read and write bytes from a serial port. Here is some example C# .NET code that uses a serial port:

```
1 // Choose the port name and the baud rate.
2 System.IO.Ports.SerialPort port = new System.IO.Ports.SerialPort("COM4", 115200);
3
4 // Connect to the port.
5 port.Open();
6
7 // Transmit two bytes on the TX line: 1, 2
8 port.Write(new byte[] { 1, 2 }, 0, 2);
9
10 // Wait for a byte to be received on the RX line.
11 int response = port.ReadByte();
12
13 // Show the user what byte was received.
14 MessageBox.Show("Received byte: " + response);
15
16 // Disconnect from the port so that other programs can use it.
17 port.Close();
```

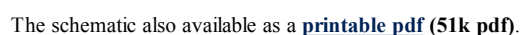
7. Ensuring a Good Radio Signal

Here are some tips for improving the quality of the radio signals sent between a pair of Wixels:

- Reduce the distance between the Wixels, if possible.
- Try different frequencies. Most Wixel apps that use the radio have a `radio_channel` parameter that determines what frequency will be used. By switching to a different channel you might be able to avoid interference from other nearby 2.4 GHz radios. You might even want to buy a spectrum analyzer such as the [Wi-Spy](#) to find out which frequencies in your area have the least activity.
- Remove objects that are very close to the Wixel's antenna. For example, if the Wixel has no header pins installed and it is resting flat on your desk, find a way to get the Wixel at least an inch or two above the desk.
- Reduce obstructions between the two Wixels. Many objects can interfere with 2.4 GHz radio waves, including walls, trees, people and anything with water in it.
- Try different Wixel orientations. The Wixel's antenna sends and receives better in some directions than others.

8. Schematic Diagram

The schematic diagram of the Wixel is shown below:



This section describes the available Wixel apps.

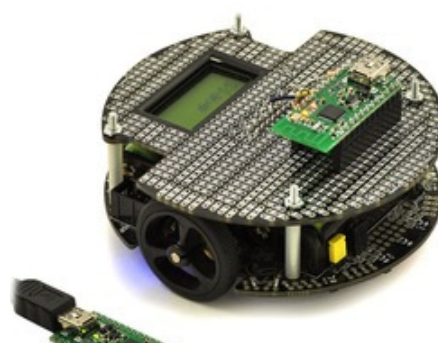
This is an example app that blinks the red LED with a configurable period. See [Section 3.d](#) for a tutorial on using this app.

Download link: [example blink led v1.0.wxl](#) (11k wxl)

Overview

This app allows you to connect two Wixels together to make a wireless, bidirectional, lossless serial link. It uses an on-air radio bit rate of 350 kbps, and is capable of carrying up to 10 KB of payload data per second and can reach a range of approximately 40 feet (under typical conditions indoors). You can also use it to turn one Wixel in to a USB-to-serial adapter.

Download [Wireless Serial App v1.1](#) (18k wxl). Open it with the Wixel Configuration Utility, choose your parameters, and then write it to two Wixels. See [Section 4](#) for more information on how this is done.



Pinout

- P1_5 = Radio Transmit Debug Signal
- P1_6 = Serial TX (0–3.3V)
- P1_7 = Serial RX (0–3.3V, not 5 V tolerant)

Description

This device appears to the USB host as a Virtual COM Port (with USB product ID 0x2200). If you are using Windows, you should see an entry labeled “Wixel” in your Device Manager in the “Ports (COM & LPT)” category while the app is running.

There are three basic serial modes that can be selected:

1. USB-to-Radio: Bytes from the USB virtual COM port get sent to the radio and vice versa.
2. UART-to-Radio: Bytes from the UART’s RX line get sent to the radio and bytes from the radio get sent to the UART’s TX line.
3. USB-to-UART: Just like a normal USB-to-Serial adapter, bytes from the virtual COM port get sent on the UART’s TX line and bytes from the UART’s RX line get sent to the virtual COM port.

You can select which serial mode you want to use by setting the `serial_mode` parameter to the appropriate number (using the numbers above). Or, you can leave the serial mode at 0 (which is the default). If the `serial_mode` is 0, then the Wixel will automatically choose a serial mode based on how it is being powered as described in the table below, and it will switch between the different serial modes on the fly.

Auto-Detect Serial Mode

Power Source	Serial Mode
USB only	USB-to-Radio
VIN only	UART-to-Radio
USB and VIN	USB-to-UART

Parameters

- **serial_mode:** Selects the serial mode (1–3, see list above) or auto-detect serial mode (0). The default is 0.
- **baud_rate:** The baud rate to use for the UART, in bits per second. The default is 9600.
- **radio_channel:** A number from 0 to 255 that determines which frequency to broadcast on. The default is 128. Wixels must be on the same channel to communicate with each other. To avoid interference, Wixels that aren’t supposed to talk to each other should be at least 2 channels away from each other. For example, you could have one pair of Wixels on channel 128 and another pair on 130.

Example Uses

1. This application can be used to make a wireless serial link between two microcontrollers, with no USB involved (except for initially configuring the Wixels). To do this, use the UART-to-Radio mode on both Wixels.
2. This application can be used to make a wireless serial link between a computer and a microcontroller. Use USB-to-Radio mode on the Wixel that is connected to the computer and use UART-to-Radio mode on the Wixel that is connected to the microcontroller. If you are powering both Wixels in the usual way, you should be able to use auto-detect serial mode (`serial_mode = 0`).
3. If you are doing option 2 above and using the the auto-detect serial mode (`serial_mode = 0`), then you have the option to (at any time) plug a USB cable directly in to the Wixel that is connected to your microcontroller to establish a more direct (wired) serial connection with the microcontroller. (You would, of course, have to tell your computer to switch to the other COM port when you do this.)

Caveats

Data will be lost if the Wixel receives bytes on the RX line faster than the radio can convey them to the other Wixel. If you have trouble, try reducing the amount of data sent to the RX line or adding delays to your microcontroller’s code.

Versions

- **[Wireless Serial App v1.1](#)** (18k wxl), released 2011-3-23: Improved the radio protocol to fix a problem in v1.0 where if one Wixel resets but the other Wixel does not, then (depending on the state of the other Wixel) there is a 50% probability that the next radio packet sent in either direction will be ignored by the receiver.
- **[Wireless Serial App v1.0](#)** (18k wxl), released 2011-3-22: Initial release.

9.c. USB-to-Serial App

Overview

This app allows you to turn a Wixel into a USB-to-TTL serial adapter capable of baud rates up to 350,000 bps. While this app does not use the radio, it has more features than the USB-to-UART



mode of the Wireless Serial App (see [Section 9.b](#)).

Installation Instructions

Download [USB-to-Serial App v1.0](#) (13k wxl). Open it with the Wixel Configuration Utility and write it to a Wixel. See [Section 4](#) for more information on how this is done.

Pinout

Pin	Function	
P1_0	$\overline{\text{DTR}}$	general purpose output pin controlled by computer
P1_1	$\overline{\text{RTS}}$	general purpose output pin controlled by computer
P1_2	$\overline{\text{DSR}}$	general purpose input pin reported to computer
P1_3	$\overline{\text{CD}}$	general purpose input pin reported to computer
P1_6	TX	transmits serial data from computer
P1_7	RX	receives data and sends it to the computer

Description

After you have loaded this app onto a Wixel, the Wixel will appear to the computer as Virtual COM Port (with USB product ID 0x2200). If you are using Windows, you should see an entry labeled “Wixel” in your Device Manager in the “Ports (COM & LPT)” category while the app is running. You can connect to this COM port using a terminal program in order to send and receive data on the TX and RX lines. Typical terminal programs will allow you to set the baud rate, parity type, and number of stop bits. Some terminal programs will allow you to use the control signals (DTR, RTS, DSR, and CD). For more information, on how to use a virtual COM port, see [Section 6](#).

This app supports all integer baud rates between 23 and 350,000 bps.

This app supports all the different types of parity: None, Odd, Even, Mark and Space.

This app supports 1 stop bit or 2 stop bits mode.

The RX line has an internal pull-up resistor, so you can leave this line disconnected.

The $\overline{\text{DSR}}$ and $\overline{\text{CD}}$ input pins have internal pull-up resistors, so when they are disconnected they will read as high (logical 0).

The $\overline{\text{DTR}}$ and $\overline{\text{RTS}}$ output pins are designed for high current (see the information on P1_0 and P1_1 in [Section 1.a](#)).

The control signals are all inverted, which means that a logical 0 corresponds to a high voltage (3.3 V) and a logical 1 corresponds to a low voltage (0 V).

This app will discard bytes received on the RX line that have framing errors or parity errors, and it will also throw out bytes if there is an RX buffer overrun. An RX buffer overrun should not happen if you are using a baud rate of 350,000 bps or less.

Example Uses

- The TX line can be used to send commands to a microcontroller or other serial device.
- The RX line can be used to receive data from a microcontroller or other serial device.
- The DTR and RTS lines are general-purpose digital outputs that can be used to control something (such as an LED) from a computer.
- The DSR and CD lines are general-purpose digital inputs that can be used to connected to a sensor or other circuit and read from a computer.

Caveats

- The CC2511’s UARTs do not actually support 1.5 stop bits, so if you try to set the number of stop bits to 1.5, this app will use 1 stop bit instead.
- The CC2511’s UARTs do not support having 2 stop bits very well, so if you set the number of stop bits to 2, this app may fail to detect framing errors that occur during the second stop bit. Also, the next byte received after the framing error occurred may be discarded even if that byte is valid. This problem only applies to receiving bytes on the RX line; this app has no problem transmitting bytes on the TX line with 2 stop bits.

9.d. I/O Repeater App

Overview

This app allows you to wirelessly extend the reach of your microcontroller’s I/O lines up to 40 feet using two or more Wixels. An input pin on one Wixel can be mapped to an output pin on another Wixel. When the input pin reads high, the output pin will be driven high (3.3 V) and when the input pin reads low, the output pin will be driven low (0 V). Each Wixel can have up to 15 input pins, 15 output pins, or a mixture of input and output pins. Each input pin can map to one or more output pins



on one or more Wixels.

Installation Instructions

Download [I/O Repeater App v1.0](#) (18k wxl). Open it with the Wixel Configuration Utility, choose your settings, and write it to two or more Wixels. See [Section 4](#) for more information on how this is done.

Description

The following 15 pins on each Wixel can be used as inputs or outputs (or be disabled):

- All the pins on Port 0: P0_0, P0_1, P0_2, P0_3, P0_4, P0_5.
- All the pins on Port 1: P1_0, P1_1, P1_2, P1_3, P1_4, P1_5, P1_6, P1_7.
- Pin P2_1 (the red LED pin).

The behavior of each pin is determined by its *link ID*, which is a parameter that you can set individually for each pin on each Wixel using the Wixel Configuration Utility. A link ID of 0 means the pin will be disabled (it will be an input but its input value will not have any effect). A negative link ID between -1 and -127 means that the pin will be a digital input and its value will be transmitted over the radio. A positive link ID between 1 and 127 means that the pin will be a digital output and its output value will be determined by the input value of the pin with the opposite (negated) link ID on another Wixel. For example, if the P1_3 pin on one Wixel has a link ID of -13, then it will be a digital input and its value will be reflected on all the output pins that have a link ID of 13 on all the other Wixels. Input pins do **not** have any effect on output pins that are on the same Wixel.

If a Wixel is running this app and has one or more pins configured to be inputs, then it will transmit a single radio packet every 10 ms (approximately) that contains input values and link IDs of all of its inputs. Any other Wixel that successfully receives this packet will process it and use it to update the state of its output pins.

Every pin configured as an input has an internal 20 kΩ pull-up resistor except P1_0 and P1_1, which float when they are inputs. This means that if you leave the input pin disconnected, it will be pulled high by default.

After you have loaded this app onto a Wixel, the Wixel will appear to the computer as Virtual COM Port (with USB product ID 0x2200). If you are using Windows, you should see an entry labeled “Wixel” in your Device Manager in the “Ports (COM & LPT)” category while this app is running. You can not send or receive data on this COM port. Its only purpose is to let the Wixel Configuration Utility easily get the Wixel in to bootloader mode.

Parameters

- **radio_channel:** A number from 0 to 255 that determines which frequency to broadcast on. The default is 128. Wixels must be on the same channel to communicate with each other. To avoid interference, Wixels that aren’t supposed to talk to each other should be at least 2 channels away from each other. For example, you could have one group of Wixels on channel 128 and another group on 130.
- **Pm_n_link:** The link ID of pin Pm_n where *m* is the port number (0–2) and *n* is the pin number (0–7).

Default Settings

The default settings are:

Pin	Link ID	Function
P0_0	-1	Input with pull-up resistor.
P2_1 (red LED pin)	1	Output linked to P0_0 on the other Wixel.

Therefore, if you load this app on to two Wixels using the default settings, they should behave as follows: If nothing is connected to either Wixel’s P0_0 line, the red LEDs on both Wixels will be on. If you connect the P0_0 line of one Wixel to GND using a wire, then you should see the red LED on the other Wixel turn off. This demonstrates the basic operation of the app.

Example Uses

- A Wixel output pin can be used to control an LED. Be sure to use an appropriate current-limiting resistor in series with the LED (e.g. 1 kΩ).
- A Wixel input pin can be used to read the state of a button or switch. Connect the button or switch between the input pin and GND, so that when the switch is open the pin will read high, and when the switch is closed the pin will read low.
- A Wixel output pin can be connected to an input pin on another microcontroller.
- A Wixel input pin can be connected to an output pin on another microcontroller. The output must not drive to a voltage higher than 3.3 V. If your microcontroller is running at 5 V you could get around this by using a diode or putting your output pin in to open-collector mode (never drive high).

Caveats

- A change on an input pin will usually be reflected on the corresponding output pin(s) within 10–100 ms, but every radio packet has a chance of being lost so there is no guaranteed latency. Therefore, this app is only suitable for very low-speed digital signals such as the signal from a pushbutton or the signal used to control an LED. This app is not suitable for PWM or RC servo signals.

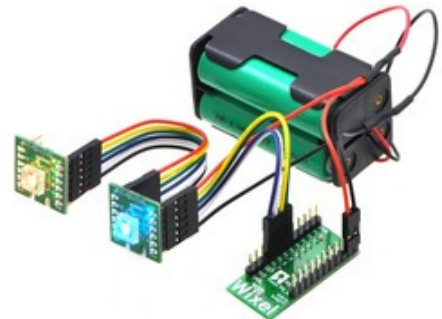
- This app uses digital I/O which means that every reading is transmitted as a 0 or 1. This app does not support analog voltages.

Caution: The Wixel's I/O lines are **not** 5V tolerant. You must use level-shifters, diodes, or voltage dividers to connect the Wixel to outputs from 5V systems. Also, avoid drawing more current from an I/O line than it can provide (see the discussion of P1_0 and P1_1 in [Section 1.a](#)). Avoid connecting multiple output pins together.

9.e. ShiftBrite App

Overview

This app allows you to wirelessly control a chain of one more [ShiftBrite](#) RGB LED modules at a distance of 40 feet or more from a PC. You will need two Wixels to set up the wireless link: one connected to the ShiftBrites, running the ShiftBrite app, and one connected to your PC with USB, running the Wireless Serial App ([Section 9.b](#)). Using a terminal or your own software, you can send a series of hex characters indicating the desired color for each module to the virtual COM port. The characters are transmitted wirelessly, received by the remote Wixel, decoded, and sent to the ShiftBrite chain, causing each module to light up with the specified color. Approximately 1000 color commands can be sent per second, allowing large displays or smooth animations.



A Wixel controlling a chain of ShiftBrites.

The ShiftBrite App is also compatible with the [ShiftBar](#), which uses the same control electronics.

Installation Instructions

Follow the instructions for the Wireless Serial App in [Section 9.b](#) to set up and test a basic wireless serial link between two Wixels. Download [ShiftBrite App v1.0](#) (18k wxl). Open it with the Wixel Configuration Utility, choose your settings, and write it to one of the Wixels. See [Section 4](#) for more information on how this is done.

Connecting the Wixel to the ShiftBrite chain

The following connections should be made between the Wixel running the ShiftBrite App and the first ShiftBrite in the chain:

Wixel	ShiftBrite	Function
P1_4	EI	Enable
P1_5	CI	Clock
P1_6	DI	Data
P1_7	LI	Latch
GND	GND	Ground

Additionally, the Wixel and ShiftBrites may share the same VIN as long as the voltage requirements for both modules are satisfied. For initial testing, you may alternatively use VALT to power the ShiftBrites from USB (see [Section 5.a](#)).

Using the ShiftBrite App

After making the correct connections and applying power, open a terminal program and connect to the COM port created by the Wixel running the Wireless Serial App. Type “fffff”. As you type the characters, they will be echoed back to your terminal. Press Enter, and the first ShiftBrite in your chain will light up in white. Then type “ff0000” and press Enter; now the first ShiftBrite should be red, and the second should be white.

To shift out multiple color commands at once, for example when you want to set the colors of the entire chain, type a series of single-color commands without pressing Enter, then press Enter once to apply them all.

Parameters

- **radio_channel:** A number from 0 to 255 that determines which frequency to broadcast on. The default is 128. Wixels must be on the same channel to communicate with each other. To avoid interference, Wixels that aren't supposed to talk to each other should be at least 2 channels away from each other. For example, you could have one pair of Wixels on channel 128 and another pair on 130.
- **input_bits:** The number of bits per channel that you would like to use to send color information. The default is 8, which allows you to send 6-digit hex values. However, the ShiftBrite supports 10 bits of resolution, so choose a value of 10 to make use of its full dynamic range. In this case, you must send 9 digits to set a color. For example, 3ff3ff3ff is the brightest possible white and 001000000 is the dimmest possible red. Other values for this parameter might be useful in special situations.
- **echo_on:** Set to 0 to disable echoing of every character sent. While echoes are useful for debugging, you might want to disable them for the highest possible speed.

Data Format

The data consists of a series of R, G, and B values, as ASCII hex strings. Each value contains from 1 to 4 characters, depending on the value of **input_bits**, specifying a number from 0 to $2^{\text{input_bits}} - 1$. When a complete set of R, G, and B values has been received, the values are multiplied or divided by the appropriate factor to match the 10-bit ShiftBrite data format and shifted out to the ShiftBrite chain. An Enter character (ASCII 10 or 13) causes the Latch pin to be toggled, instantly setting each ShiftBrite to its new color.

Tips

- The yellow LED turns on when the first Enter is received, which might be useful for debugging your wireless connection.
- You can use [wires with pre-crimped terminals](#) and [crimp connector housings](#) to make a custom cable between the Wixel and the ShiftBrite chain.

10. Writing Your Own Wixel App

10. Troubleshooting

Trouble getting the Wixel Configuration Utility to recognize your Wixel

To force your Wixel to go in to bootloader mode so that the Wixel Configuration Utility will recognize it, follow these steps:

1. Power off your Wixel by removing power or put it in to reset mode by driving the RST pin low. If you choose to power off the Wixel, then be aware that Wixel can be inadvertently powered through any IO line if is connected to a voltage source that is driving high.

10.a. Getting Started in Windows

To get started developing your own Wixel Apps using Windows as the development platform, we recommend that you download and install the Pololu Wixel Development Bundle: [wixel dev bundle 110322.exe](#) (9393k exe). This bundle contains four things:

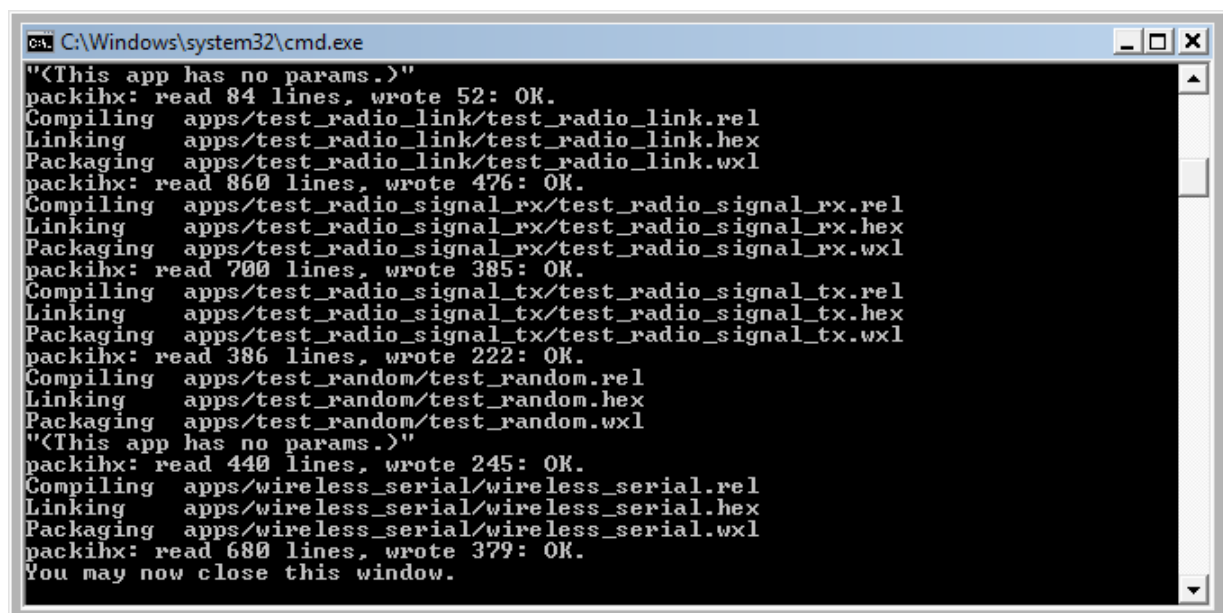
- **Wixel SDK**: Contains source code (for libraries and applications) that will help you develop your own applications for the Wixel.
- **SDCC – Small Device C Compiler**: A free compiler that is used to compile the code in the Wixel SDK.
- Pololu GNU Build Utilities: Windows versions of some open-source utilities that are required by the SDK.
- **Notepad++**: A free text editor which is convenient to use when editing Wixel code.

10.b. Compiling an Example App

After you have installed all the necessary software (either from the Wixel Development Bundle or from other sources), you should try compiling an example app and loading it onto a Wixel to make sure that your system is set up correctly.

Compiling Apps

Go to the folder where you installed the Wixel SDK (C:\wixel-sdk is the default) and run `make_all.bat`. If everything is installed correctly, the output of that batch file should look something like this:



```

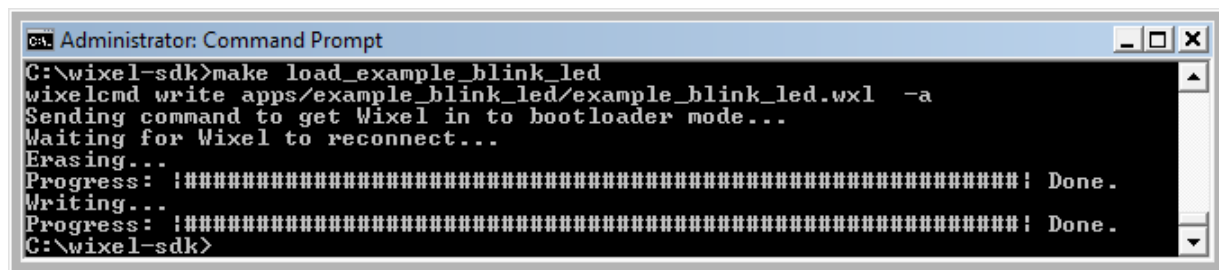
C:\Windows\system32\cmd.exe
"<This app has no params.>"
packihx: read 84 lines, wrote 52: OK.
Compiling apps/test_radio_link/test_radio_link.rel
Linking apps/test_radio_link/test_radio_link.hex
Packaging apps/test_radio_link/test_radio_link.wxl
packihx: read 860 lines, wrote 476: OK.
Compiling apps/test_radio_signal_rx/test_radio_signal_rx.rel
Linking apps/test_radio_signal_rx/test_radio_signal_rx.hex
Packaging apps/test_radio_signal_rx/test_radio_signal_rx.wxl
packihx: read 700 lines, wrote 385: OK.
Compiling apps/test_radio_signal_tx/test_radio_signal_tx.rel
Linking apps/test_radio_signal_tx/test_radio_signal_tx.hex
Packaging apps/test_radio_signal_tx/test_radio_signal_tx.wxl
packihx: read 386 lines, wrote 222: OK.
Compiling apps/test_random/test_random.rel
Linking apps/test_random/test_random.hex
Packaging apps/test_random/test_random.wxl
"<This app has no params.>"
packihx: read 440 lines, wrote 245: OK.
Compiling apps/wireless_serial/wireless_serial.rel
Linking apps/wireless_serial/wireless_serial.hex
Packaging apps/wireless_serial/wireless_serial.wxl
packihx: read 680 lines, wrote 379: OK.
You may now close this window.
  
```

You have now built all the apps in the Wixel SDK and all the libraries that they depend on. Every subfolder of the `apps` folder should now have a WXL file in it. This is a compiled app which can be loaded onto Wixels using the Wixel Configuration Utility or the Wixel Command-Line Utility (WixelCmd).

Loading an App onto Wixels from the Command Line

While you are developing an app, it can be useful to have a way to load an app onto the target Wixel from the command line. You can do this with the Makefile: plug a Wixel in to your computer, open up a command prompt, navigate to the `wixel-sdk` folder, and type `make load_example_blink_led`. This should compile the `example_blink_led` app (if it is out of date) and then invoke the Wixel Command-Line

Utility (WixelCmd) in order to load the app onto all Wixels connected to your computer. The output should look something like this:



```

C:\wixel-sdk>make load_example_blink_led
wixelcmd write apps/example_blink_led/example_blink_led.wxl -a
Sending command to get Wixel in to bootloader mode...
Waiting for Wixel to reconnect...
Erasing...
Progress: #####! Done.
Writing...
Progress: #####! Done.
C:\wixel-sdk>

```

The example blink LED application should be running now. The Wixel's yellow LED should be off, and the red LED should be blinking. If that is the case, then congratulations! You have successfully compiled a program and loaded it onto the Wixel.

You can also specify settings at the command line. Try running this command:

```
make load_example_blink_led S="blink_period_ms=100"
```

You should now see the LED blinking 5 times faster than it was before. To specify multiple settings, put other settings inside the quotes and separate all settings with spaces.

Opening an App with the Wixel Configuration Utility

You can also open the app in the Wixel Configuration Utility if you want to. Try running this command:

```
make open_example_blink_led
```

The Wixel Configuration Utility should start running and open the example_blink_led app.

Creating Your Own Apps

Now that you know how to compile apps and quickly load them onto the Wixel to test them, you are ready to develop your own Wixel apps. You can either modify one of the existing apps in the SDK or create your own app. To create your own app, simply copy one of the existing app folders and change the name. You do **not** need to modify the Makefile when you create a new app; the Makefile will automatically detect the app. When you are developing your app, you can use all of the commands above except you should replace “example_blink_led” with the name of your app (the name of the subfolder in the apps folder).

10.c. Sharing Your App with the Wixel Community

Preparing your app for the community

Please make sure that you have set up the following sections in your WXL file:

- **description** – this should fully describe what your app does or link to a web site with a complete description. Since other people will be trying this on their Wixels, make sure to bring our attention to anything that may cause damage or incompatibility. Most importantly, *any IO ports used as outputs should be explicitly listed* in the description.
- **license** – this specifies the terms under which others may distribute copies or modifications of your app. We recommend and use the MIT license, which is simple and allows widespread use of your code. Note that if you have used the Wixel SDK to build your app, you need to include our license, though you may add your own restrictions if necessary. We recommend using the following template, which will make it easy for us to share your app or to include it in the Wixel SDK:

```

Copyright (c) 2011 <YOUR NAME>. Documentation for this app is available at:

http://<YOUR SITE>/

Copyright (c) 2011 Pololu Corporation. For more information, see

http://www.pololu.com/
http://forum.pololu.com/

Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without

```

Announcing your app on the Wixel forum

When you are ready for the community to try out a version of your app, you may announce it on [the Wixel forum](http://forum.pololu.com/), attaching your WXL file. It is helpful to paste a copy of your **description** into your post so that we can see what your app is about before downloading it.

Contributing your source code

Sharing your source code greatly increases the value of your app, since other members of the community will be able to adapt it to their own projects or learn from your code. You might also have made changes to the common sections of the Wixel SDK that you would like us to incorporate, or you might have discovered and want to share your fixes. For all of these reasons, we are maintaining the [Pololu Wixel SDK Repository on GitHub](#) as a central location for sharing Wixel code. You can easily use this site to fork your own version of the Wixel SDK, show us what you are working on, submit changes, and so on.

GitHub uses the [Git Version Control System](#) for tracking versions and branches of code, so we highly recommend learning how to use Git if you want to share your code with the Wixel community or even just to get the latest version for your own use. Teaching Git is beyond the scope of this User's Guide, but there are many good tutorials available on the web. For example, this [Git tutorial](#) provided by GitHub focuses on Windows installation and configuration, but it should be useful to Linux and Mac users as well.

For the rest of this section, we will assume that you are comfortable with Git.

If you have been following the instructions in this document, your code is located within the `wixel-sdk` folder along with the rest of the Wixel SDK. This folder is actually a *clone* of our Git repository, which means that it includes history and version information as well as a link back to our repository (a *remote*), within the folder `.git`. As long as you have not modified the contents of `.git`, you should be able to use Git to *commit* your changes, *pull* updates from us, and *push* your changes to a public repository. To make sure that Git is working correctly, try typing the following command at the command prompt from within the `wixel-sdk` folder:

```
git log
```

This should display a list of recent commits in the repository. As an alternative, you can use the program `gitk` to view a much more detailed, graphical representation of the log. After reviewing the log, you can add and commit all of your new files:

```
git add <list of new files...>
git commit -a
```

When you run the commit, you will be shown a list of new or modified files and prompted for a commit message. Please check this over carefully and write a good description of your changes, because it is very hard to change your commits after you make them public.

In some cases, you might want to start over with a fresh copy of the Wixel SDK, which you can make by typing

```
git clone -o pololu git://github.com/pololu/wixel-sdk.git
```

into a command prompt. At any time, you can download the latest updates from us and merge them into your code by typing the following command:

```
git pull pololu
```

Initially, you will commit your changes to your local repository, but when you are ready to publish them, you should set up your own repository on GitHub to make them accessible to the community. To do this, sign up for a GitHub account and fork our repository into a repository on your account. Please do this by clicking the “fork” button at the top of [our repository](#) so that it will be connected within the GitHub system – this way we will see your changes and can easily incorporate them into the SDK. After making your fork, you can push all of the changes from your local repository to GitHub with

```
git remote add myrepo <GitHub URL>
git push myrepo
```

where `<GitHub URL>` is the SSH or HTTP URL shown on your GitHub page.

Finally, you will probably want to post a link to your GitHub repository on [the Wixel forum](#). This will help others find out about it and give you feedback on your contribution.

10.d. USB Configurations Recognized by the Wixel Configuration Software

The Wixel Configuration Utility and the Wixel Command-Line Utility can currently only recognize certain Wixel apps. If your app is not recognized, then users of your app will not easily be able to reconfigure it after they have loaded it onto their Wixel (they will have to get their Wixel in to bootloader mode using some other method).

To be recognized, your app must implement a particular USB interface. Specifically, it must implement a USB CDC ACM virtual COM port with vendor ID 0x1FFB and product ID 0x2200. When the USB host sends a command to your app that sets the baud rate to 333, your app should enter bootloader mode. Your app should work with the `wixel_serial.inf` driver which is installed along with the other Windows software (see [Section 3.a](#)). The easiest way

to write an app that meets these requirements is to use the `usb_cdc_acm.lib` library in the Wixel SDK.

In the future we will add support for other types of USB interfaces if there is a need for it. Please [contact us](#) if you would like to use other types of interfaces in your apps.